# Sqirvy-llm

**VERSION 0.0.1**

I was working out some devops for a project, and I needed a simple way to make queries to LLM providers for use in a Go command line program. I didn't want to have to copypasta from a web app or a Python script. This project is an attempt to create the simplest possible Go api for making queries to LLM providers. I wanted to use Go because it is convenient to build binaries for Linux, Windows and MacOS.

GitHub Repo

## API Library

This is the interface you would use to make queries to LLM providers in Go.

Most of the code was generated using Aider and the claude-3-sonnet-20240229 model. I had to do several iterations with Aider and some manual editing to get the exact code layout I wanted.

The API is in directory pkg/api. It is a very simple interface that allows you to query a provider with a prompt and get a response. It supports Anthropic, Gemini, and OpenAI providers through the 'client' interface. Here is an example of how to use the API in a command line program. Examples for the other providers are in the 'cmd' directory.

- Making a query to a provider
  - Create a new client for the provider you want to use
    - api.NewClient(api.)
    - anthropic, gemini or openai
  - Make the query with a prompt, the model name, and any options (nothing supported yet). You can request the results to be plain text or JSON
    - client.QueryText(prompt, model string, options Options) (string, error)
    - client.QueryJSON(prompt string, model string, options Options) (string, error)
  - Get the response
  - Handle any errors

```
package main

import (
    "fmt"
    "log"

    api "sqirvyllm/pkg/api"
)

func main() {
    // Create a new Anthropic client
    client, err := api.NewClient(api.Anthropic)
    if err != nil {
        log.Fatalf("Failed to create client: %v", err)
    }

    // Make the query with a prompt, the model name, and any options
(nothing supported yet)
    response, err := client.QueryText("say hello world", "claude-3-sonnet-
20240229", api.Options{})
    if err != nil {
        log.Fatalf("Query failed: %v", err)
    }

    fmt.Println("Response:", response)
}
```

## Example Usage

### Build The Executables

- the build system uses 'make'
- 'make' can be run from top level or from the cmd or web directories
- build (or default)
  - build the binaries for the cmd and web directories
  - builds cmd binaries for the current OS platform and also Linux, MacOS, and Windows versions
  - after a `make build` or `make test`, the binaries will be in the `bin` directory
- test
  - run the tests
- clean
  - remove the binaries

### Fixed Arguments

Use these as a model if you want to create a command line app with fixed arguments. For example, this could be useful in DevOps if you want to pipeline some query output during a code review. Executables are in the 'bin' directory. source is in the 'cmd' directory.

- cmd/Anthropic
  - a hello world query to Anthropic
- cmd/Gemini
  - a hello world query to Gemini
- cmd/OpenAI
  - a hello world query to OpenAI

### cmd/sqirvy-query : Chainable Command Line Interface To Query Models

The cmd/sqirvy-query directory contains a command line program that allows you to send prompts to specified AI models. Stand alone, it lets you pipe a prompt or specify files to use to compose a prompt. It supports chaining multiple prompts together and can read prompts from standard input (stdin) and/or files, concatenate them, and send the combined prompt to the specified AI model.

**Usage: sqirvy-query [options] files...**

- concatenates prompt from stdin and/or files and sends it to the specified AI model
- defaults to Anthropic claude-3-5-sonnet-latest model if no model is specified
- example: pipe a prompt to sqirvy-query with the default claude-3-5-sonnet-latest model
  - echo "say hello world" | sqirvy-query
- example: read a prompt from a file and pipe it to sqirvy-query with the o1-mini model
  - sqirvy-query -m o1-mini prompt.txt
- Options:
  - -h print this help message
  - -m AI model to use (default: claude-3-5-sonnet-latest)
- Supported models:
- claude-3-5-haiku-latest
- claude-3-5-sonnet-latest
- claude-3-opus-latest
- gemini-1.5-flash
- gemini-1.5-pro
- gemini-2.0-flash-exp
- gpt-4-turbo
- gpt-4o
- gpt-4o-mini
- o1-mini

**Chaining Prompts**

```
#!/bin/bash
query="../../bin/sqirvy-query"

# this example uses the system.md prompt by default and
# then uses a chain of prompts to generate the code.py file
# system.md   : a system prompt for software engineers
#             : this is the default system prompt used by sqirvy-query if
it is in the local directory
# describe.md : a general description of well formed python code
#             : uses gemini-1.5-flash model for this query
# generate.md : a description of the specific code to generate
#             : uses claude-3-5-sonnet-latest model for this query
# tee code.py : outputs the result to the file code.py to the terminal and
```

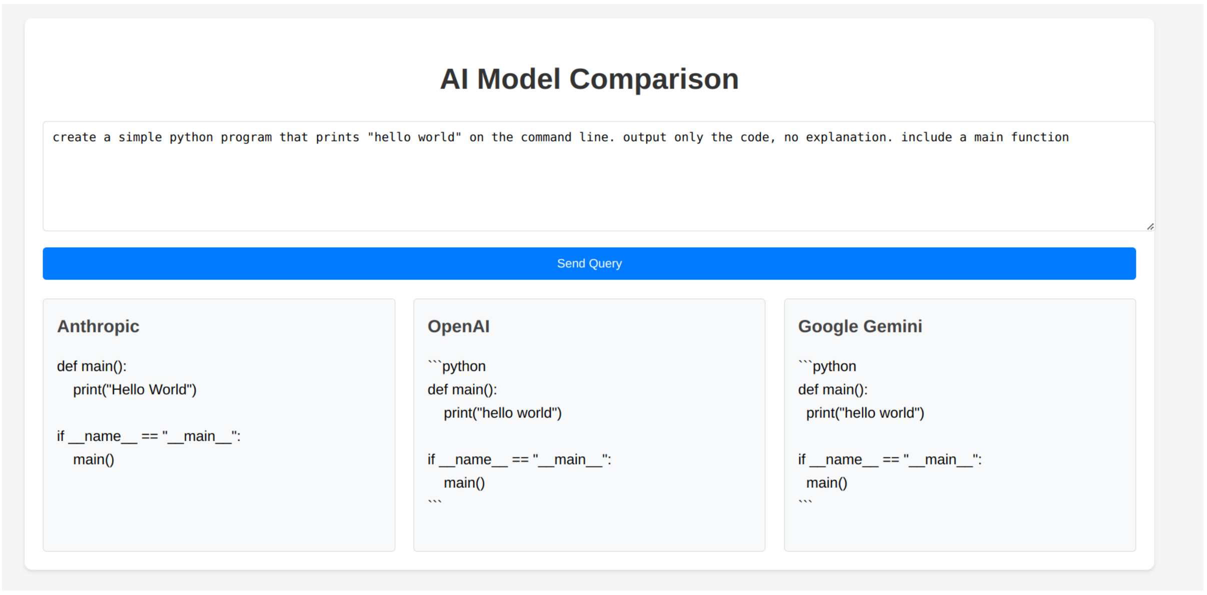
```
to the file system


# create the prompt files then pipe them to the queries
$query -m gemini-1.5-flash describe.md |\
$query -m claude-3-5-sonnet-latest generate.md |\
tee code.py
```

### web/sqirvy-web

A simple web app that allows you to query all three providers in parallel and compare the results.

- cd into "web/sqirvy-web"
- go run .
- it will start a web server on port 8080.



```
create a simple python program that prints "hello world" on the command line. output only the code, no explanation. include a main function
```

Anthropic

```
def main():
    print("Hello World")

if __name__ == "__main__":
    main()
```

OpenAI

~~~
```python
def main():
    print("hello world")

if __name__ == "__main__":
    main()
```
~~~

Google Gemini

~~~
```python
def main():
  print("hello world")

if __name__ == "__main__":
  main()
```
~~~

The code for the web app was generated using Aider and the claude-3-sonnet-20240229 model.

## What Client API's Were Used

### Anthropic

- github.com/anthropics/anthropic-sdk-go
- this api is a Go native client for the Anthropic API
- this api is the one recommeded by Anthropic for Go.
- It's in alpha now but seems to work without problems for these use cases.
- **The Anthropic sdk default to "ANTHROPIC_API_KEY" environment variable to authenticate**

### Gemini

- "github.com/google/generative-ai-go/genai"
- "google.golang.org/api/option"
- this is the official Go client for the Gemini API supported by Google
- **The Gemini API requires a "GEMINI_API_KEY" environment variable to authenticate**

## OpenAI

- OpenAI API
- Since there did not seem to be an official Go native API for OpenAI, I used the OpenAI REST API directly with the "net/http" package.
- **The OpenAI API requires a "OPENAI_API_KEY" environment variable to authenticate**